ClearContracts *Lightpaper*

An Introduction to Clear Contracts

Templated smart contract library APIs

Join our Discord | Twitter | LinkedIn



Our Mission

The mission of Clear Contracts is to lower the barrier to entry to create, manage, and execute smart contracts. Smart contracts have the ability to make existing systems more optimized and equitable.

Access to technology with this potential should not be limited to a small group of highly technical individuals. Rather, it should be accessible to as many people as possible.

Smart contracts are ready to disrupt existing systems based of their ability to utilize blockchain technology to track and automate transactions and processes in conjunction with legal documentation.

Clear Contracts is at the forefront of providing a non technical solution for this technology to be implemented into the industries who would benefit from it the most.

The Opportunity

The opportunity in front of us comes to light because of three facts of the blockchain industry.

Writing high assurance smart contracts is difficult, time consuming, and expensive. It is extremely inefficient for each individual project in the Cardano ecosystem to devote time and money to develop much of the same code.

The technical barrier to entry to create and interact with smart contracts on the cardano blockchain is too high. Blockchain concepts like Dapps are not inclusive to all if only a select few individuals possess the technical ability to create and interact with them.

Obtaining contract audits is expensive and labor intensive. Small projects with the desire to create verifiably secure dapps have no affordable way to access audited contracts.

The Solution

Clear Contracts provides a non technical solution so that anyone can create, manage, and execute high assurance smart contracts.

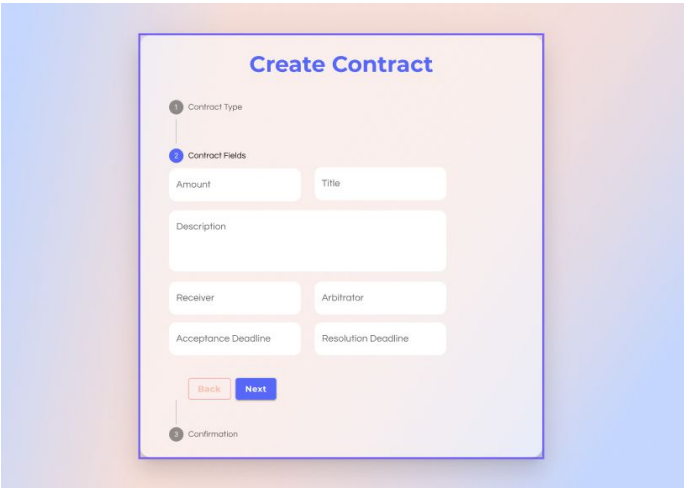
We provide a comprehensive solution that enables our users to interact with audited smart contract templates out of the box.

We turn complicated, audited haskell code...

```haskell
{-# LANGUAGE TemplateHaskell #-}

module Week08.Lens where

import Control.Lens

newtype Company = Company {_staff :: [Person]} deriving Show

data Person  = Person
    { _name    :: String
    , _address :: Address
    } deriving Show

newtype Address = Address {_city :: String} deriving Show

alejandro, lars :: Person
alejandro = Person
  {  _name    = "Alejandro"
  ,  _address = Address {_city = "Zacateca"}
  }
lars = Person
  {  _name    = "Lars"
  ,  _address = Address {_city = "Regensburg"}
  }

iohk :: Company
iohk = Company { _staff = [alejandro, lars] }

goTo :: String -> Company -> Company
goTo there c = c {_staff = map movePerson (_staff c)}
  where
    movePerson p = p {_address = (_address p) {_city = there}}
```

```
Withdraw (Wallet 1) (Wallet 2) 1 2
SetPrice (Wallet 1) (Wallet 2) 1
BuyTokens (Wallet 2) (Wallet 2) 0
Start (Wallet 2)
Start (Wallet 1)
AddTokens (Wallet 1) (Wallet 2) 6
BuyTokens (Wallet 1) (Wallet 1) 1
Prelude Plutus.Contract.Test.ContractModel Test.QuickCheck Spec.Model > :t nextState
nextState :: ContractModel state => Action state -> Spec state ()
Prelude Plutus.Contract.Test.ContractModel Test.QuickCheck Spec.Model > :t view

<interactive>:1:1: error: Variable not in scope: view
Prelude Plutus.Contract.Test.ContractModel Test.QuickCheck Spec.Model > import Control.Lens
Prelude Plutus.Contract.Test.ContractModel Test.QuickCheck Control.Lens Spec.Model > :t view
view
  :: mtl-2.2.2:Control.Monad.Reader.Class.MonadReader s m =>
     Getting a s a -> m a
Prelude Plutus.Contract.Test.ContractModel Test.QuickCheck Control.Lens Spec.Model > :t perform
perform
  :: ContractModel state =>
     HandleFun state
     -> ModelState state
     -> Action state
     -> Plutus.Trace.Emulator.EmulatorTrace ()
Prelude Plutus.Contract.Test.ContractModel Test.QuickCheck Control.Lens Spec.Model > :t propRunActions
propRunActions              propRunActions_             property
propRunActionsWithOptions  properFraction             propertyForAllShrinkShow
Prelude Plutus.Contract.Test.ContractModel Test.QuickCheck Control.Lens Spec.Model > :t propRunActionsWithOptions
propRunActionsWithOptions
  :: ContractModel state =>
     Plutus.Contract.Test.CheckOptions
     -> [ContractInstanceSpec state]
```

into a simple online form.



Technical Developments:

Simple Escrow Smart Contract based off captured user interactions

One of our initial contracts is a simple escrow smart contract that takes the user's desired inputs, deploys a contract to the blockchain, and manages the eventual facilitation of funds to the correct party.

This contract can be applied across a variety of industries because it provides an easy way for any two parties to engage in a trustworthy and transparent agreement. In order to drive more adoption of this contract is capable of using any token including stablecoins, tokens for DAOs, ADA, etc...

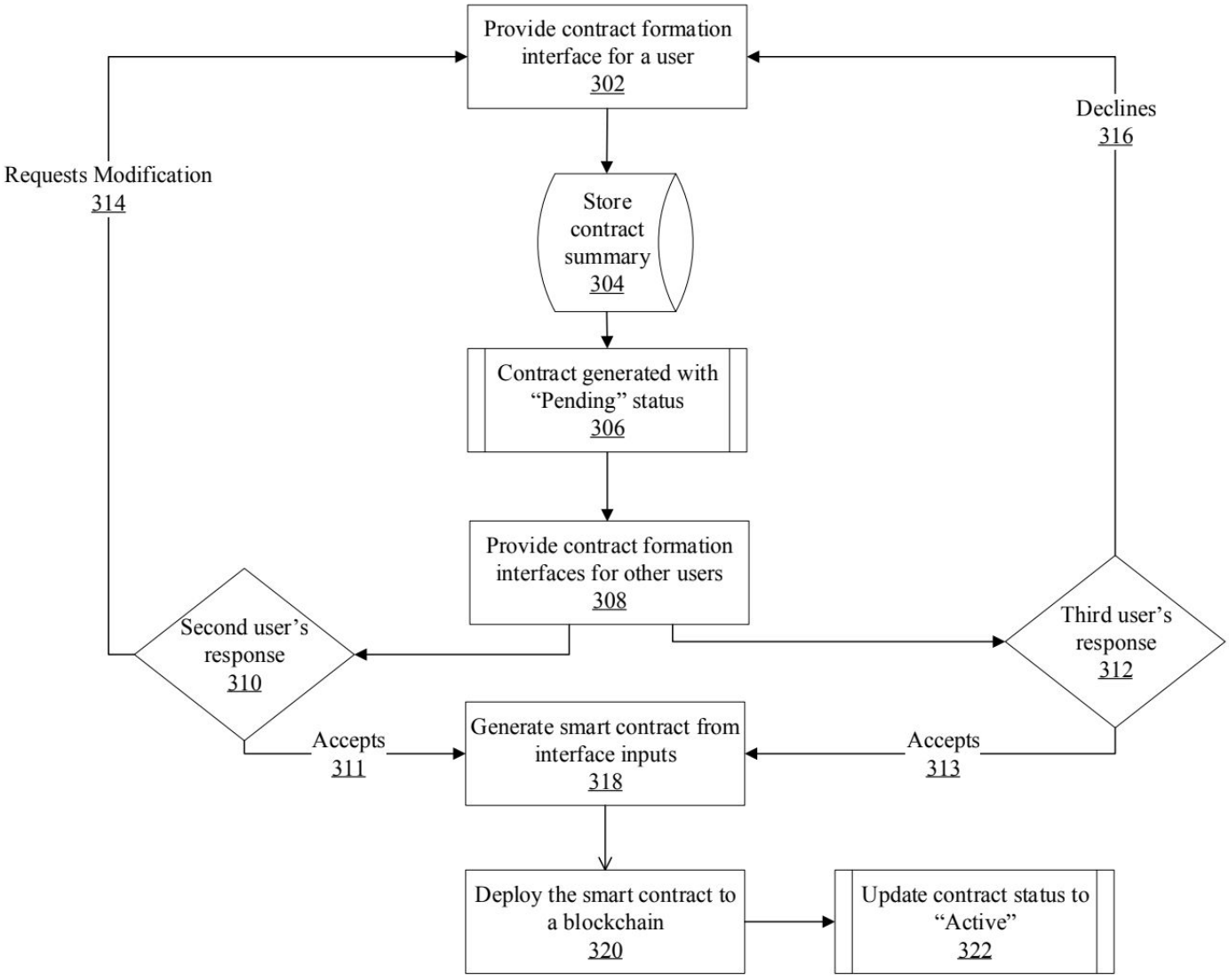


Fig. 1: Simple Escrow Smart Contract deployment logic

Technical Developments:

Simple Escrow Smart Contract based off captured user interactions Cont.

This flow chart details how the logic of the escrow contract ensures that funds are allocated to the correct party to complete execution of the contract based off captured user interactions. In the event of a disagreement between the parties, an arbitrator is assigned at the beginning of the process to resolve disputes.

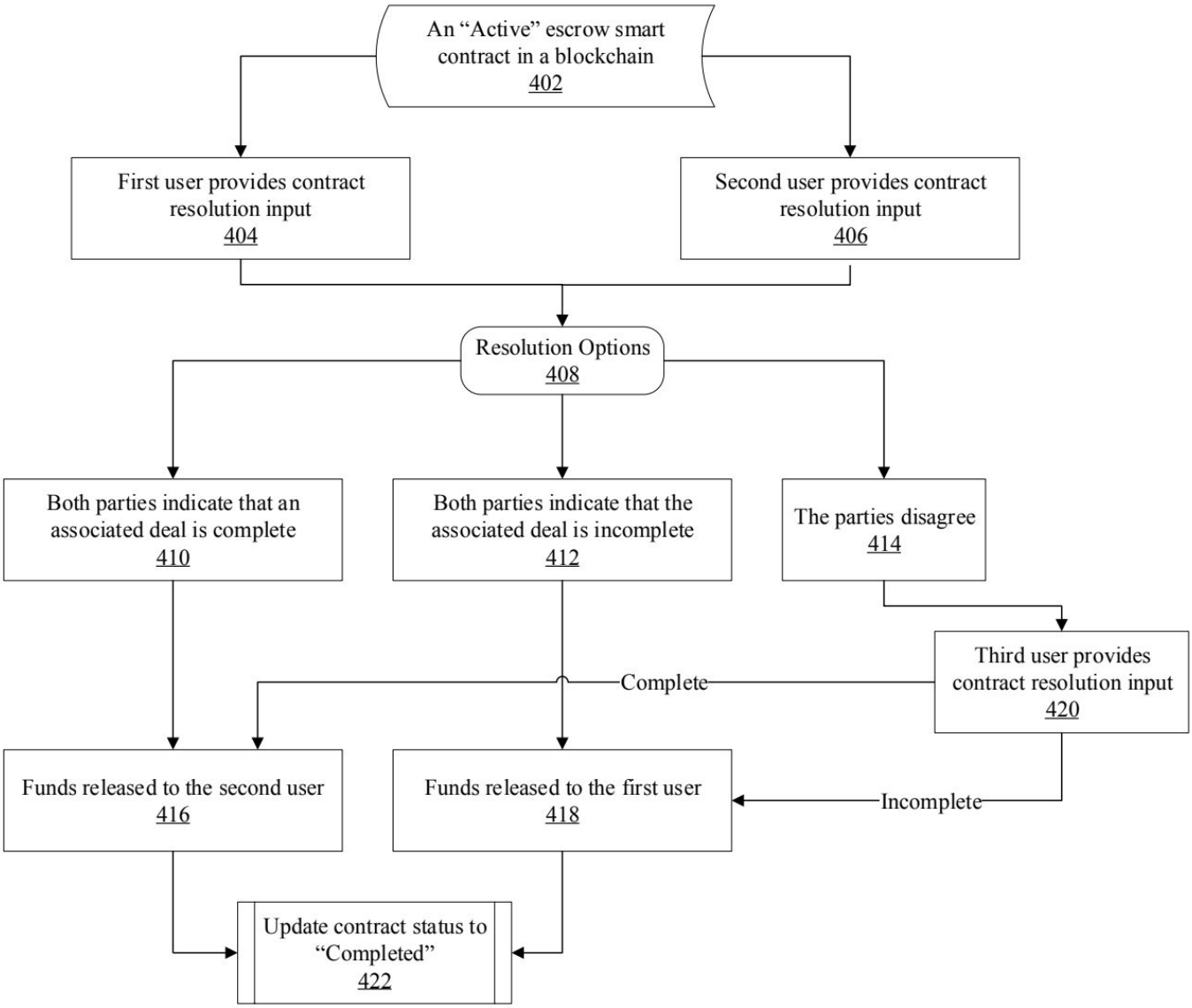


Fig. 2: Simple Escrow Smart Contract fund allocation logic

Roadmap

Beta product is a simple smart contract management platform and serves as a proof of concept to templetize plutus contracts.

Phase 1: Launch Beta product

Our API will serve as the on ramp for 3rd party applications to integrate high assurance smart contracts.

Phase 2: API Creation

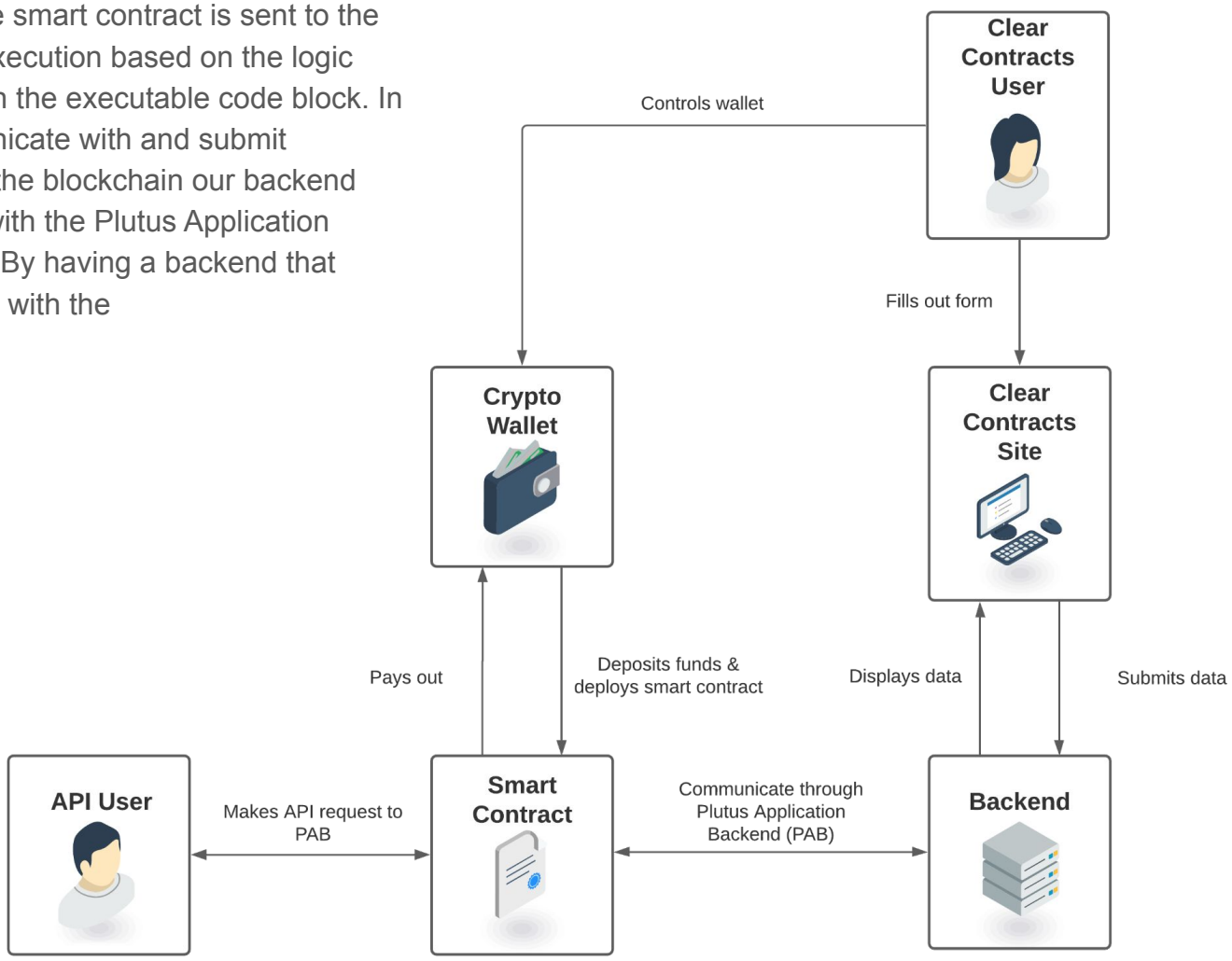
We will expand smart contract library to create contracts that satisfy use cases in collaboration with the community.

Phase 3: Expand Smart Contract Library

Platform Architecture

Our platform and services serve as direct access points to create, manage, and execute smart contracts on the Cardano Blockchain. In order to create a contract, a user must fill out a form online. The form inputs get collected and stored by our internal database. When all parties involved in the agreement accept their roles in the contract, our platform automatically populates the corresponding smart contract template and the smart contract is sent to the blockchain for execution based on the logic embedded within the executable code block. In order to communicate with and submit transactions on the blockchain our backend communicates with the Plutus Application Backend (PAB). By having a backend that works in tandem with the

PAB we are able to provide a unique solution that enables on chain actions to be triggered from our web application and services.



ClearContracts

Bringing smart contract technology to the real world